Business Intelligence & Big Data Analytics Journal

M.Sc Part I Computer Science

### Yash Salvi 548

### March 22, 2023



### R.J. College of Arts, Science & Commerce Business Intelligence & Big Data Analytics

### Seat number: 548



**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

## Ghatkopar (W), Mumbai-400 086

**CERTIFICATE**

M.Sc Part I Computer Science 2022-2023

This is to certify that **Yash Sanjay Salvi** of M.Sc Prat I (Sem-II) Computer Science, Seat No **548** of satisfactorily completed the practicals of **ABUSINESS INTELLI- GENCE & BIG DATA ANALYTICS (PAPER I)** during the academic year **2022 - 2023** as specified by the **MUMBAI UNIVERSITY**.

No. of Experiments completed **10** out of **10 Sign of Incharge:**

#### Date: March 22, 2023 Seat Number: 548

#### Sign of Examiner:

#### Date: March 22, 2023 Course Co-ordinator

**Contents**

[Practical No: 1](#_bookmark0) 1

[Practical No: 2](#_bookmark1) 2

[Practical No: 3](#_bookmark2) 5

[Practical No: 4](#_bookmark3) 6

[Practical No: 5](#_bookmark4) 8

[Practical No: 6](#_bookmark5) 9

[Practical No: 7](#_bookmark6) 12

[Practical No: 8](#_bookmark7) 14

[Practical No: 9](#_bookmark8) 17

**List of Figures**

# Practical No: 1

## Aim: Switching ON and OFF LED using Push button with Arduino UNO. Practical:

1. Download the latest stable release of Apache Hadoop from the official website.
2. Extract the downloaded file to a directory on your system.
3. Set the HADOOP\_HOME environment variable to the location where Hadoop is installed using the following command: export HADOOP\_HOME=/path/to/hadoop
4. Set the JAVA\_HOME environment variable to the location where Java is installed using the following command: export JAVA\_HOME=/path/to/java
5. Add the bin directory of Hadoop to your PATH environment variable using the following command:

export PATH=$PATH:$HADOOP\_HOME/bin

1. Configure the Hadoop cluster by editing the hadoop-env.sh file located in the $HADOOP\_HOME/etc/hadoop/ directory. Set the JAVA\_HOME and other configuration parameters as required.
2. Configure the Hadoop core-site.xml file by setting the Hadoop filesystem URI and other properties as required. This file is located in the $HADOOP\_HOME/etc/hadoop/ directory.
3. Configure the Hadoop hdfs-site.xml file by setting the replication factor and block size for HDFS as required. This file is also located in the $HADOOP\_HOME/etc/hadoop/ directory.
4. Start the Hadoop daemons using the following command: $HADOOP\_HOME/sbin/start-all.sh
5. Verify that Hadoop is running by accessing the web interface at http://localhost:50070/.

# Practical No: 2

## Aim: Implementing Map-Reduce Program for Word Count problem. Description:

Here’s the step-by-step procedure for implementing a Map-Reduce program for the Word Count problem:

1. Map function: The map function takes an input record (a line of text) and outputs a list of key-value pairs, where the key is a word in the line and the value is 1.

1

$\operatorname{Map}(key, value):$

words $\gets$ Split value on whitespace for each word in words:

EmitIntermediate(word, 1)

2

3

4

1. Shuffle and sort: The intermediate key-value pairs are grouped by key and sorted by key, so that all occurrences of a particular word are grouped together. This step is done automatically by the Map- Reduce framework.
2. Reduce function: The reduce function takes an intermediate key and a list of values and outputs a single key-value pair, where the key is the word and the value is the total count of that word across all input records. Here’s the reduce function in LaTeX code:

1

$\operatorname{Reduce}(key, values):$ count $\gets$ 0

for each value in values:

count $\gets$ count + value Emit(key, count)

2

3

4

5

1. Driver program: The driver program sets up the Map-Reduce job by specifying the input and output paths, the map and reduce functions, and any other configuration options. Here’s an example of a driver program in LaTeX code:

1

$\operatorname{main}():$

job $\gets$ new MapReduceJob() job.setInputPath("/path/to/input") job.setOutputPath("/path/to/output") job.setMapper(Map)

2

3

4

5

6

job.setReducer(Reduce) job.run()

7

This is a basic outline of how a Map-Reduce program for the Word Count problem works. Note that there are many variations and optimizations of this basic approach, depending on the specifics of the input data and the desired output.

## Code:

1

**import java.io.IOException**; **import java.util.StringTokenizer**;

**import org.apache.hadoop.conf.Configuration**; **import org.apache.hadoop.fs.Path**;

**import org.apache.hadoop.io.IntWritable**; **import org.apache.hadoop.io.Text**;

**import org.apache.hadoop.mapreduce.Job**; **import org.apache.hadoop.mapreduce.Mapper**; **import org.apache.hadoop.mapreduce.Reducer**;

**import org.apache.hadoop.mapreduce.lib.input.FileInputFormat**; **import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat**;

**public class WordCount** {

**public static class TokenizerMapper**

**extends** Mapper<Object, Text, Text, IntWritable>{

**private final static** IntWritable one = **new** IntWritable(1);

**private** Text word = **new** Text();

**public void** map(Object key, Text value, Context context

) **throws** IOException, InterruptedException { StringTokenizer itr = **new** StringTokenizer(value.toString()); **while** (itr.hasMoreTokens()) {

word.set(itr.nextToken()); context.write(word, one);

}

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

|  |  |  |  |
| --- | --- | --- | --- |
| 32  33  34  35 | **public static class IntSumReducer**  **extends** Reducer<Text,IntWritable,Text,IntWritable> {  **private** IntWritable result = **new** IntWritable(); | | |
| 36 | **public void** reduce(Text key, Iterable<IntWritable> values, | |  |
| 37 | Context context | |  |
| 38 | ) **throws** IOException, InterruptedException | | { |
| 39 | **int** sum = 0; | |  |
| 40 | **for** (IntWritable val : values) { | |  |
| 41 | sum += val.get(); | |  |
| 42 | } | |  |
| 43 | result.set(sum); | |  |
| 44 | context.write(key, result); | |  |
| 45 | } | |  |
| 46 | } | |  |
| 47 |  | |  |
| 48 | **public static void** main(String[] args) **throws** Exception { | |  |
| 49 |  | Configuration conf = **new** Configuration(); | |
| 50 |  | Job job = Job.getInstance(conf, "word count"); | |
| 51 |  | job.setJarByClass(WordCount.class); | |
| 52 |  | job.setMapperClass(TokenizerMapper.class); | |
| 53 |  | job.setCombinerClass(IntSumReducer.class); | |
| 54 |  | job.setReducerClass(IntSumReducer.class); | |
| 55 |  | job.setOutputKeyClass(Text.class); | |
| 56 |  | job.setOutputValueClass(IntWritable.class); | |
| 57 |  | FileInputFormat.addInputPath(job, **new** Path(args[0])); | |
| 58 |  | FileOutputFormat.setOutputPath(job, **new** Path(args[1])); | |
| 59 |  | System.exit(job.waitForCompletion(**true**) ? 0 : 1); | |
| 60 |  | } | |
| 61 | } |  | |

## Output :

The seven-segment will display the numbers starting from 0 to 9 after every 1000 ms as the delay de is declared as 1000.

# Practical No: 3

## Practical

Pig is a high-level scripting language used for analyzing large datasets. It provides a simple syntax for performing data transformations and analysis. Here’s an example Pig script for solving counting problems: Assume we have a dataset containing the following information about customers:

1

(**John**, **30**, **M**)

(**Mary**, **25**, **F**)

(**Tom**, **40**, **M**)

(**Emily**, **35**, **F**)

2

3

4

5

The goal is to count the number of male and female customers in the dataset. Here’s the Pig script to accomplish this task:

1

*-- Load the dataset*

customers = **LOAD** 'path/to/customers.csv' **USING** PigStorage(',')

**AS** (name: chararray, age: int, gender: chararray);

*-- Group the data by gender*

gender\\_group = **GROUP** customers **BY** gender;

*-- Count the number of records in each group*

gender\\_count = FOREACH gender\\_group GENERATE **group**, **COUNT**(customers);

*-- Store the result*

STORE gender\\_count **INTO** 'path/to/output';

2

3

4

5

6

7

8

9

10

11

12

13

In the above script, we first load the dataset using the LOAD command and define the schema of the data using the AS keyword. Then, we group the data by the gender field using the GROUP command. Next, we use the FOREACH command to count the number of records in each group using the COUNT function. Finally, we store the result using the STORE command.

The output will be stored in a file at the specified output path, containing the following records:

1

(**M**,**2**)

(**F**,**2**)

2

This shows that there are 2 male and 2 female customers in the dataset.

# Practical No: 4

## Aim: Install HBase and use the HBase Data model Store and retrieve data. Overview:

1. Start by creating a table in HBase, specifying its name and column families. A column family is a

group of related columns that are stored together. For example, you might have a column family called "personal" for storing personal information, and another called "professional" for storing professional information.

1. To store data in HBase, you need to create a Put object for each row of data you want to insert. The Put object specifies the row key, the column family, the column qualifier (which identifies the specific column within the column family), and the value.
2. Once you have created your Put objects, you can use the HTable.put() method to insert them into HBase.
3. To retrieve data from HBase, you can create a Get object for each row of data you want to retrieve. The Get object specifies the row key, and can also specify the column family and column qualifier if you want to retrieve a specific column.
4. You can then use the HTable.get() method to retrieve the data. This method returns a Result object, which contains the data for the specified row.
5. To extract the data from the Result object, you can use the getXXX() methods, where XXX is the data type of the column. For example, if you have a column called "age" that contains an integer, you can use the Result.getInt() method to retrieve the value.
6. You can also use scan operations to retrieve multiple rows of data at once. A scan operation returns a ResultScanner object, which allows you to iterate over the results.
7. When you are done working with your table, you should close the HTable object to free up resources.

## Code:

1

//Start HBase hbase shell

//HBase Commands status version,

table\_help whoami

//Data Definition Language create ‘employee’, ’Name’, ’ID’,

2

3

4

5

6

7

8

9

’Designation’, ’Salary’, ’Department’

//Verify created table list

//Disable single table disable ‘employee’ scan ‘employee’

//or

is\_disable ‘employee’

//Disable multiple tables disable\_all ‘e.\*’

// Enabling table enable‘employee’

//Or is\_enabled'employee'

//create new table

create‘student’, ‘name’, ‘age’, ‘course’

put ‘student’, ‘sharath’, ‘name:fullname’, ‘sharathkumar’ put ‘student’, ‘sharath’, ‘age:presentage’, ‘24’

put ‘student’, ‘sharath’, ‘course:pursuing’, ‘Hadoop’ put ‘student’, ‘shashank’, ‘name:fullname’, ‘shashank R put ‘student’, ‘shashank’, ‘age:presentage’, ‘23’

put ‘student’, ‘shashank’, ‘course:pursuing’, ‘Java’

//Get Information

get ‘student’, ‘shashank’ get ‘student’, ‘sharath’

get ‘student’, ‘sharath’, ‘course’ get ‘student’, ‘shashank’, ‘course’ get ‘student’, ‘sharath’, ‘name’

//Scan

scan ‘student’

//Count

Count ‘student’

//Alter

alter ‘student’, NAME=>’name’, VERSIONS=>5

put ‘student’, ‘shashank’, ‘name:fullname’, ‘shashank Rao’ scan ‘student’

//Delete

delete ‘student’, ‘shashank’, ‘name:fullname’

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

## Output :

LCD screen display the message “Hello world”.

# Practical No: 5

## Aim: : Install Hive and use Hive Create and store structured databases. Code:

1

cat > /home/cloudera/employee.txt 1~Sachine~Pune~Product Engineering~100000~Big Data 2~Gaurav~Banglore~Sales~90000~CRM 3~Manish~Chennai~Recruiter~125000~HR 4~Bhushan~Hyderabad~Developer~50000~BFSI cat /home/cloudera/employee.txt

sudo -u hdfs hadoop fs -put /home/cloudera/employee.txt /inputdir hdfs dfs -ls / hdfs dfs -ls /inputdirectory

hadoop fs -cat /inputdirectory/employee.txt hive

show databases;

create database organization; show databases;

use organization; show tables;

hive> create table employee(

* id int,
* name string,
* city string,
* department string,
* salary int,
* domain string)
* row format delimited
* fields terminated by '~'; show tables;

**select** \* from employee; show tables;

load data inpath '/inputdir/employee.txt' overwrite into table employee; show tables;

**select** \* from employee;

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

## Output :

# Practical No: 6

## Aim:Write a program to construct different types of k-shingles for a given doc- ument.

## Overall:

1. Install and configure Hive: First, you need to install and configure Hive on your system. You can follow the instructions provided in the Hive documentation for your specific environment to do this.
2. Define your data schema: Next, you need to define the schema for your data. This involves specifying the data types and column names for each field in your dataset. You can define your schema using the CREATE TABLE statement in Hive.
3. Create a Hive table: Once you have defined your schema, you can create a Hive table to store your data. You can do this using the CREATE TABLE statement in Hive, and specifying the table name and the schema you defined in the previous step.
4. Load data into your table: Once your table is created, you can load data into it. You can do this using the LOAD DATA statement in Hive, which allows you to specify the source of your data and the destination table.
5. Query your table: With your data loaded into your table, you can now query it using the SELECT statement in Hive. This allows you to retrieve specific subsets of your data based on criteria such as date ranges, locations, or other filters.
6. Analyze your data: Finally, with your data stored in Hive, you can use Hive’s built-in data analysis functions to gain insights into your data. For example, you can use Hive’s GROUP BY and ORDER BY clauses to group and sort your data, or use Hive’s JOIN statement to combine data from multiple tables.

## Code:

1

install.packages("tm") require("tm") install.packages("devtools") readinteger < - function () {

n < -readline(prompt = "Enter value of k-1: ") k < -as.integer(n)

u1 < -readLines("c:/msc/r-corpus/File1.txt")

Shingle < -0

i < -0

while (i < nchar(u1) - k + 1) {

2

3

4

5

6

7

8

9

10

11

Shingle[i] < -substr(u1, start = i, stop = i + k) print(Shingle[i])

i = i + 1

}

}

if (interactive()) readinteger()

12

13

14

15

16

#### Output :

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | |
| 1 | * **if**(interactive()) | | readinteger() |
| 2 | Enter value of k-1: | | 2 |
| 3 | character(0) | |  |
| 4 | [1] | "thi" | |
| 5 | [1] | "his" | |
| 6 | [1] | "is " | |
| 7 | [1] | "s i" | |
| 8 | [1] | " is" | |
| 9 | [1] | "is " | |
| 10 | [1] | "s a" | |
| 11 | [1] | " a " | |
| 12 | [1] | "a t" | |
| 13 | [1] | " te" | |
| 14 | [1] | "tex" | |
| 15 | [1] | "ext" | |
| 16 | [1] | "xt." | |
| 17 | [1] | "t. " | |
| 18 | [1] | ". i" | |
| 19 | [1] | " it" | |
| 20 | [1] | "it " | |
| 21 | [1] | "t i" | |
| 22 | [1] | " is" | |
| 23 | [1] | "is " | |
| 24 | [1] | "s s" | |
| 25 | [1] | " sh" | |
| 26 | [1] | "sho" | |
| 27 | [1] | "hor" | |
| 28 | [1] | "ort" | |
| 29 | [1] | "rt," | |
| 30 | [1] | "t, " | |
| 31 | [1] | ", a" | |
| 32 | [1] | " an" | |

|  |  |  |
| --- | --- | --- |
| 33 | [1] | "and" |
| 34 | [1] | "nd " |
| 35 | [1] | "d i" |
| 36 | [1] | " it" |
| 37 | [1] | "it " |
| 38 | [1] | "t i" |
| 39 | [1] | " is" |
| 40 | [1] | "is " |
| 41 | [1] | "s o" |
| 42 | [1] | " on" |
| 43 | [1] | "one" |
| 44 | [1] | "ne " |
| 45 | [1] | "e l" |
| 46 | [1] | " li" |
| 47 | [1] | "lin" |
| 48 | [1] | "ine" |
| 49 | [1] | "ne." |
| 50 | [1] | "e. " |
| 51 | [1] | ". i" |
| 52 | [1] | " it" |
| 53 | [1] | "it " |
| 54 | [1] | "t i" |
| 55 | .... |  |
| 56 |  |  |
|  |  |  |

# Practical No: 7

## Aim: Write a program for measuring similarity among documents and detecting passages which have been reused.

## Description:

1. Install and load required packages: tm, ggplot2, textreuse, and devtools.
2. Create a Corpus object my.corpus from a directory of text files using Corpus(DirSource()).
3. Remove English stopwords from my.corpus using tm\_map().
4. Create a TermDocumentMatrix object my.tdm from my.corpus using TermDocumentMatrix().
5. Create a DocumentTermMatrix object my.dtm from my.corpus using DocumentTermMatrix() with term frequency-inverse document frequency weighting and stopword removal.
6. Convert my.tdm to a data frame my.df using as.data.frame(inspect(my.tdm)).
7. Scale my.df using scale().
8. Compute the Euclidean distance matrix d from the scaled data using dist().
9. Cluster the documents using hierarchical clustering with Ward’s linkage method using hclust().
10. Plot the dendrogram of the hierarchical clustering using plot().

## Code:

1

install.packages("tm") require("tm") install.packages("ggplot2") install.packages("textreuse") install.packages("devtools")

my.corpus <- Corpus(DirSource("C:/MSC Notes/r-corpus"))

my.corpus <- tm\_map(my.corpus, removeWords, stopwords("english")) my.tdm <- TermDocumentMatrix(my.corpus)

my.dtm <- DocumentTermMatrix(my.corpus,

control = list(weighting = weightTfIdf, stopwords = **TRUE**)) my.df <- as.data.frame(inspect(my.tdm))

my.df.scale <- scale(my.df)

d <- dist(my.df.scale, method = "euclidean") fit <- hclust(d, method = "ward")

plot(fit)

2

3

4

5

6

7

8

9

10

11

12

13

14

15

#### Output :

1

<TermDocumentMatrix (terms: 69, documents: 6)>> Non-/sparse entries: 97/317

Sparsity : 77% Maximal term length: 12

Weighting : term frequency (tf) Docs

.

...

2

3

4

5

6

7

8

# Practical No: 8

## Aim: Write a program to compute the n- moment for a given stream where n is given.

## Overall

:

Here are the steps to write a program to compute the N-moment for a given stream:

1. Define the input: The input to the program is a stream of numbers and the value of N, which represents the order of the moment that needs to be computed.
2. Define the variables: Define the variables needed to compute the moment. We need a variable to store the current sum, a variable to store the number of items in the stream, and a variable to store the current value of the moment.
3. Read the input stream: Read the input stream one value at a time. For each value, add it to the current sum, increment the count of items, and calculate the current value of the moment using the formula: *moment* = *sum*(*itemN* )*/count*.
4. Output the result: Once all the values in the stream have been processed, output the final value of the moment.

## Code:

1

**package pract**;

**import java.util.ArrayList**; **import java.util.Arrays**;

**import java.util.stream.Stream**; **public class Pract** {

**public static void** main(String[] args) {

**int** n = 15;

String stream[] = { "a",

"b",

"c",

"b",

"d",

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28 };

29

"a",

"c",

"d",

"a",

"b",

"d",

"c",

"a",

"a",

"b"

30 **int** zeroMoment = 0, firstMoment = 0, secondMoment = 0, count = 1, flag = 0;

31

32 ArrayList < Integer > arrlist = **new** ArrayList();

33

34 System.out.println("ArrayList element are:");

35

36 **for** (**int** i = 0; i < 5; i++) {

37 System.out.println(stream[i] + " ");

38 }

39 Arrays.sort(stream);

40

41 **for** (**int** i = 1; i < n; i++) {

42 **if** (stream[i] == stream[i - 1]) {

43 count++;

44 } **else** {

45 arrlist.add(count);

46 count = 1;

47 }

48 }

49 arrlist.add(count);

50

51 zeroMoment = arrlist.size();

52

53 System.out.println("Value of zero moment is:" + zeroMoment);

54

55 **for** (**int** i = 0; i < arrlist.size(); i++) {

56 firstMoment += arrlist.get(i);

57 }

58 System.out.println("Values of firstMoment:" + firstMoment);

59

**for** (**int** i = 0; i < arrlist.size(); i++) {

**int** j = arrlist.get(i); secondMoment += (j \* j);

}

System.out.println("value of secont moment is:" + secondMoment);

}

}

60

61

62

63

64

65

66

67

## Output :

When the program is loaded in arduino UNO, Servo motor starts rotating.

# Practical No: 9

## Aim: Write a program to demonstrate the Alon-Matias-Szegedy Algorithm for second moments.

## Overview

The Alon-Matias-Szegedy (AMS) algorithm is a randomized algorithm that estimates the second moments of a dataset. It is commonly used in data stream algorithms and is useful for analyzing large datasets that cannot be stored in memory. The basic idea behind the algorithm is to hash the data in a way that preserves the second moments and then estimate the second moments from the hashed values. Here are the steps to implement the AMS algorithm:

1. Step 1: Hash the data Choose a family of pairwise-independent hash functions, *h* : *U →* 1*,* 1, where U is the universe of the input dataset. Apply each hash function h to each element of the dataset to obtain a hashed value. Store the hashed values in a table.
2. Step 2: Compute the first moment Compute the sum of the hashed values. This gives an estimate of the first moment of the dataset.
3. Step 3: Compute the second moment For each element in the dataset, compute the sum of the hashed values for all pairs of elements that contain that element. Compute the average of the sum of the hashed values for each element. This gives an estimate of the second moment of the dataset.
4. Step 4: Return the estimate Return the estimate of the second moment obtained in step 3. Note that the AMS algorithm may not give an exact estimate of the second moment, but it has been shown to provide a good approximation with high probability. The accuracy of the estimate depends on the number of hash functions used and the size of the dataset. By using more hash functions or increasing the size of the table, the accuracy of the estimate can be improved.

## Code:

1

**import java.io.\***; **import java.util.\***; **class AMSA** {

**public static int** findCharCount(String stream, **char** XE, **int** random, **int** n) {

**int** countoccurance = 0;

**for** (**int** i = random; i < n; i++) {

**if** (stream.charAt(i) == XE) { countoccurance++;

}

}

**return** countoccurance;

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43 }

}

**public static int** estimateValue(**int** XV1, **int** n) {

**int** ExpValue;

ExpValue = n \* (2 \* XV1 - 1);

**return** ExpValue;

}

**public static void** main(String args[]) {

**int** n = 15;

String stream = "abcbdacdabdcaab";

**int** random1 = 3, random2 = 8, random3 = 13;

**char** XE1, XE2, XE3;

**int** XV1, XV2, XV3;

**int** ExpValuXE1, ExpValuXE2, ExpValuXE3;

**int** apprSecondMomentValue;

XE1 = stream.charAt(random1 - 1); XE2 = stream.charAt(random2 - 1); XE3 = stream.charAt(random3 - 1);

XV1 = findCharCount(stream, XE1, random1 - 1, n); XV2 = findCharCount(stream, XE2, random2 - 1, n); XV3 = findCharCount(stream, XE3, random3 - 1, n);

System.out.println(XE1 + "=" + XV1 + " " + XE2 + "=" + XV2 + " " + XE3 + "=" + XV3);

ExpValuXE1 = estimateValue(XV1, n); ExpValuXE2 = estimateValue(XV2, n); ExpValuXE3 = estimateValue(XV3, n);

System.out.println("Expected value for" + XE1 + " is::" + ExpValuXE1); System.out.println("Expected value for" + XE2 + " is::" + ExpValuXE2); System.out.println("Expected value for" + XE3 + " is::" + ExpValuXE3); apprSecondMomentValue = (ExpValuXE1 + ExpValuXE2 + ExpValuXE3) / 3; System.out.println("approximate second moment value using alon-matis-szegedy is::"

+ apprSecondMomentValue);

}

#### Output:

1

yashsalvi @Nightfury16 ~/Downloads/MSc/BIG D % java prac9.java c=3 d=2 a=2

Expected value forc is::75 Expected value ford is::45 Expected value fora is::45

2

3

4

5

6

approximate second moment value using alon-matis-szegedy is::55